Apple Assembly Line

$1.80

Volume 4 -- Issue 8 May, 1984

In This Issue...

This month we are beginning a series of articles describing a double-precision decimal arithmetic package for Applesoft. Imagine 18-digit arithmetic with none of the screwy rounding errors we are used to seeing in Applesoft's binary arithmetic.

You will also find quick looks at the new Apple //c and a forthcoming set of revised ROMs for the //e. We finally have the solution to a three-year-old mystery! You old-timers might remember that in August of 1981 we published a peculiar little "what does this code do?" item from John Broderick. Well he has revealed answer at long last.

Oops!

There are a couple of bugs in the Intellec Hex Converter we published last month. To correct the program you should delete line 2240 (the INY) and add a LDA #0 at line 2285. That will take care of it! Our thanks to Chaim Palman, of Calcomp, for pointing out the problems.

Random Numbers for Applesoft.....Bob Sander-Cederlof

The RND function in Applesoft is faulty, and many periodicals have loudly proclaimed its faults. "Call APPLE", Jan 83, pages 29-34, tells them in "RND is Fatally Flawed", and presents an alternative routine which can be called with the USR function.

First, the flaws: 1) the initialization code fails to preset all five bytes of the seed value (only the first four of five are loaded); 2) the RND code uses a poor algorithm, and depends on "tweaks" to make the numbers more random; 3) the RND code does not properly implement the algorithm it appears to be aiming at.

BAD INITIALIZATION. The initialization code is at $F150 in the Applesoft ROMs. This loop moves the CHRGET subroutine down to $B1-C8, and is also supposed to copy the random number seed into $C9-CD. The last byte does not get copied, due to a bug. Changing $F151 from $1C to $1D would fix it. Most of us don't really care about this bug, because we are trying to get random numbers for games and the like, and the more random the better: not copying the last byte could make the numbers generated a little more random from one run to the next. However, some applications in simulation programs require REPEATABLE sequences of random numbers, so the effect of model changes can be seen independent of the random number generator.

POOR ALGORITHM. Most generators use an algorithm which makes the next random number by multiplying the previous one by a constant, and adding another constant. The result is reduced by dividing by a third constant and saving the remainder as the next random number. More on this later. The proper choice of the three constants is critical. I am not sure whether the Applesoft authors just made poor choices, or whether the bugs mentioned below drove them to tweaking. Tweaking the generated value is often thought to produce even more random results. In fact, according to authorities like Donald Knuth, they almost always ruin the generator. Applesoft tweaks the generated value by reversing the middle two bytes of the 32-bit value. Guess what: it ruins the generator, assuming it was good to start with.

BUGGY ALGORITHM. The congruency algorithm described in words above will only work properly when integer arithmetic is used. Applesoft uses floating point arithmetic. Further, Applesoft arithmetic routines expect five-byte operands. For some reason the constants used in RND are only four bytes long each. It appears that the exponents may have been omitted, in the expectation that integer arithmetic was going to be used. You can see the code for RND at $EFAE.

If you want to see some non-random features using RND, type in and RUN the following program:

```
10 HGR:HCOLOR=3
20 X=RND(1)*280:Y=RND(1)*160
30 HPLOT X,Y
40 GO TO 20
```

S-C Macro Assembler Version 1.0.....$80
S-C Macro Assembler Version 1.1.....$92.50
Version 1.1 Update.....$12.50
Source Code for Version 1.1 (on two disk sides).....$100
Full Screen Editor for S-C Macro (with complete source code).....$49
S-C Cross Reference Utility (without source code).....$20
S-C Cross Reference Utility (with complete source code).....$50
DISASM Dis-Assembler (RAK-Ware).....$30
Source Code for DISASM.....additional $30

S-C Word Processor (with complete source code).....$50
Double Precision Floating Point for Applesoft (with source code).....$50
S-C Documentor (complete commented source code of Applesoft ROMs).....$50
Source Code of //e CX & F8 ROMs on disk.....$15

(All source code is formatted for S-C Macro Assembler Version 1.1. Other assemblers require some effort to convert file type and edit directives.)

AAL Quarterly Disks.....each $15
Each disk contains all the source code from three issues of "Apple Assembly Line", to save you lots of typing and testing time.

QD#1: Oct-Dec 1980	QD#2: Jan-Mar 1981	QD#3: Apr-Jun 1981
QD#4: Jul-Sep 1981	QD#5: Oct-Dec 1981	QD#6: Jan-Mar 1982
QD#7: Apr-Jun 1982	QD#8: Jul-Sep 1982	QD#9: Oct-Dec 1982
QD#10: Jan-Mar 1983	QD#11: Apr-Jun 1983	QD#12: Jul-Sep 1983
QD#13: Oct-Dec 1983	QD#14: Jan-Mar 1984	

OBJ.APWRT][F (Don Lancaster, Synergetics).....$30
Quick-Trace (Anthro-Digital).....(reg. $50) $45
Visible Computer: 6502 (Software Masters).....(reg. $50) $45
ES-CAPE: Extended S-C Applesoft Program Editor.....$60
Amper-Magic (Anthro-Digital).....(reg. $75) $65
Amper-Magic Volume 2 (Anthro-Digital).....(reg. $35) $30
Routine Machine (Southwestern Data Systems).....(reg. $64.95) $60
FLASH! Integer BASIC Compiler (Laumer Research).....$79
Fontrix (Data Transforms).....$75
Aztec C Compiler System (Manx Software).....(reg. $199) $180
IACcalc Spreadsheet Program.....(reg. $84.95) $75
The one we use every day. It's better than Visicalc!
Locksmith 5.0 (Omega MicroWare).....(reg. $99.95) $90

Blank Diskettes (Verbatim).....2.50 each, or package of 20 for $45
(Premium quality, single-sided, double density, with hub rings)
Vinyl disk pages, 6"x8.5", hold two disks each.....10 for $6
Diskette Mailing Protectors (hold 1 or 2 disks).....40 cents each or $25 per 100
These are cardboard folders designed to fit into 6"X9" Envelopes.
Envelopes for Diskette Mailers..... 6 cents each
ZIF Game Socket Extender (Ohm Electronics) $20

Books, Books, Books.....compare our discount prices!
"Apple][Circuit Description", Gayler.....($22.95) $21
"Understanding the Apple II", Sather.....($22.95) $21
"Enhancing Your Apple II, vol. 1", Lancaster.....($15.95) $15
Second edition, with //e information.
"Incredible Secret Money Machine", Lancaster.....($7.95) $7
"Beneath Apple DOS", Worth & Lechner.....($19.95) $18
"Bag of Tricks", Worth & Lechner, with diskette.....($39.95) $36
"Assembly Lines: The Book", Roger Wagner.....($19.95) $18
"What's Where in the Apple", Second Edition.....($24.95) $23
"What's Where Guide" (updates first edition).....($9.95) $9
"6502 Assembly Language Programming", Leventhal.....($18.95) $18
"6502 Subroutines", Leventhal.....($17.95) $17
"Real Time Programming -- Neglected Topics", Foster.....($9.95) $9

We have small quantities of other great books, call for titles & prices.
Add $1.50 per book for US shipping. Foreign orders add postage needed.

*** S-C SOFTWARE, P. O. BOX 280300, Dallas, TX 75228 ***
*** (214) 324-2050 ***
*** We accept Master Card, VISA and American Express ***

You will see the Hi-Res screen being sprinkled with dots. After about seven minutes, but long before the screen is full, new dots stop appearing. RND has looped, and is replotting the same sequence of numbers. Another test disclosed that the repetition starts at the 37,758th "random" number.

Mathematicians have developed many sophisticated tests for random number generators, but Applesoft fails even these simple ones! Depending on the starting value, you can get the Applesoft generator in a loop. You never get anywhere near the theoretically possible 4 billion different values.

The Call APPLE article proposes a new algorithm. It comes with impressive claims and credentials, but I have not found it to be better than a properly implemented congruential algorithm. The algorithm multiplies the previous seed by 8192, and takes the remainder after dividing by 67099547. This is a congruency algorithm:

$$X(n+1) = (a * X(n) + c) \bmod m$$

with a=8192, c=0, m=67099547

I re-implemented the Call APPLE algorithm, and my listing follows. The Call APPLE version would not quite fit in page 3, but mine does with a little room to spare. I also dug into some other references and came up with another algorithm, from Knuth. It is also a congruency, but with a=314159269, c=907633386, and m=2^32. This turns out to be easier to compute, and according to Knuth it should be "better". "Better" is in quotes because it is really hard to pin down what are the most important properties. Anyway this one should have very good characteristics.

The RND function does three different things, depending on the argument. You write something like R=RND(X). If X=0, you get the same number as the previous use of RND produced. If X<0, the absolute value of X becomes the new seed value. This allows you to control the sequence when you wish, and also to randomize it somewhat by using a "random" seed. If X>0, you get the next random number. The value will always be a positive number less than 1. If you want to generate a number in a range, you multiply by the width of the range and add the starting value. For example, to generate a random integer between 1 and 10:

```
R = INT( RND(1)*10 ) + 1
```

The programs I have written build a little on the options available with RND. They all begin with a little routine which hooks in the USR vector. After executing this, you can write R=USR(X), in other words substitute USR(X) anywhere you would have used RND(X). But I have added, following the Call APPLE article, the option to automatically generate integers in a range based at 0. If 0<X<2, you will get the next random fraction. If X is 2 or greater than 2, you will get a random integer between 0 and X-1. Thus you can make a random integer between 1 and 10 like this:

```
R = USR(10) + 1
```

as well as with:

```
R = INT (USR(1)*10) + 1
```

I wrote a third program which makes a 16-bit random value. This one uses the seed at $4E and $4F which the Apple increments continuously whenever the standard monitor input loop is waiting for an input keystroke. Integer BASIC uses this seed, and as a result is quite valuable in writing games. My new program gives you all the options stated above, and is significantly quicker than any of the others. It uses a=19125, c=13843, and m=2^16 in a standard congruency algorithm.

If you are seriously interested in random numbers, you need to read and study Donald Knuth. Volume 2 of his series "The Art of Computer Programming" is called "Seminumerical Algorithms". Chapter 3, pages 1-160, is all about random numbers. (There is only one other chapter in this volume, all about arithmetic in nearly 300 pages!) Knuth started the series back in the 60's, with the goal of seven volumes covering most of what programmers do. He finished the first three by 1972, went back and revised the first one, and then evidently got sidetracked into typesetting (several books around a typesetting language he calls "Tex").

Speaking of being sidetracked...!

Knuth ends his chapter with a list of four rules for selecting a, c, and m for congruency algorithms. Let me summarize those rules here:

1. The number m is conveniently taken as the word size. In Applesoft, the floating point mantissa is 32 bits; hence, I chose m=2^32.

2. If m is a power of 2 (and mine is), pick "a" so that "a mod 8 = 5". This, together with the rules on choosing c below, ensure that all m values will produced before the series repeats.

3. Pick "a" between m/100 and m-sqrt(m). The binary digits should NOT have a simple, regular pattern. Knuth recommends taking some haphazard constant, such as a=3131492621.

4. "c" should be odd, and preferable near "m*.2113248654".

Now for the program listings.

The first listing is for my rendition of Call APPLE's algorithm. Lines 1220-1280 link in the USR vector. Lines 1370-1450 branch according to the value of the argument of the USR function. If the argument is negative, lines 1550-1620 set up its absolute value as the new seed. If the argument is

zero, the old seed is used without change, lines 1420-1450. If positive non-zero, lines 1470-1490 set up the argument as the RANGE.

Lines 1640-1690 calculate the new seed, which will be 8192 times the old seed, modulo 67099547. 8192 is 2^13, so we can multiply be 13 left shifts. After each shift, if the result is bigger than 67099547, we subtract that value and keep the remainder. The final result will be some number smaller than 67099547.

Lines 1700-1770 save the new seed, and then divide it by 67099547 to get a fraction for the USR function result. Lines 1780-1860 check the initial argument to see if you wanted a fraction between 0 and 1, or an integer between 0 and arg-1. If the latter, the fraction is multiplied by the range and reduced to an integer.

The subroutine named MODULO subtracts 67099547 from the seed value if it would leave a positive remainder, and then renormalizes the result into floating point.

Line 2270 defines the initial seed after loading the program to be 1.0. If you want some other seed, change this line or be sure to seed it with R=USR(-seed) in your Applesoft program.

```
                    1000 *--------------------------------
                    1010 *SAVE S.USRND S-C
                    1020 *--------------------------------
                    1030 *       FROM CALL APPLE, JAN 1983, PAGE 29-34
                    1040 *--------------------------------
                    1050         .OR $300
                    1060 *       .TF B.USRND
                    1070 *--------------------------------
E82E-               1080 NORMALIZE.FAC       .EQ $E82E
E97F-               1090 FMUL.FAC.BY.YA      .EQ $E97F
E9E3-               1100 LOAD.ARG.FROM.YA    .EQ $E9E3
EA5C-               1110 FDIV.ARG.BY.YA      .EQ $EA5C
EAF9-               1120 LOAD.FAC.FROM.YA    .EQ $EAF9
EB2B-               1130 STORE.FAC.AT.YX.ROUNDED .EQ $EB2B
EB66-               1140 COPY.FAC.TO.ARG     .EQ $EB66
EC23-               1150 AS.INT              .EQ $EC23
                    1160 *--------------------------------
000A-               1170 USER.VECTOR         .EQ $0A THRU $0C
009D-               1180 FAC                 .EQ $9D THRU $A2
00A2-               1190 FAC.SIGN            .EQ $A2
00A5-               1200 CNTR                .EQ $A5
                    1210 *--------------------------------
0300- A9 4C         1220 LINK    LDA #$4C    "JMP" OPCODE
0302- 85 0A         1230         STA USER.VECTOR
0304- A9 0D         1240         LDA #RANDOM
0306- 85 0B         1250         STA USER.VECTOR+1
0308- A9 03         1260         LDA /RANDOM
030A- 85 0C         1270         STA USER.VECTOR+2
030C- 60            1280         RTS
                    1290 *--------------------------------
                    1300 *       R = USR (X)
                    1310 *       IF X < 0 THEN RESEED WITH ABS(X)
                    1320 *       IF X = 0 THEN R = REPEAT OF PREVIOUS VALUE
                    1330 *       IF 0 < X < 2 THEN GENERATE NEXT SEED AND RETURN
                    1340 *                         0 <= R < 1
                    1350 *       IF X >= 2 THEN R = INT(RND*X)
                    1360 *--------------------------------
                    1370 RANDOM
030D- A5 A2         1380         LDA FAC.SIGN CHECK FOR RESEEDING
030F- 30 1C         1390         BMI .2       ...YES
0311- A5 9D         1400         LDA FAC      CHECK FOR X=0
0313- D0 0A         1410         BNE .1       ...NO, X=RANGE
0315- A9 AE         1420         LDA #SEED
0317- A0 03         1430         LDY /SEED
0319- 20 E3 E9      1440         JSR LOAD.ARG.FROM.YA
031C- 4C 55 03      1450         JMP .5
```

```
                      1460 *---X --> RANGE-----------------
031F- A2 A9           1470 .1     LDX #RANGE
0321- A0 03           1480        LDY /RANGE
0323- 20 2B EB        1490        JSR STORE.FAC.AT.YX.ROUNDED   $EB2B
                      1500 *---SEED --> FAC---------------
0326- A9 AE           1510        LDA #SEED
0328- A0 03           1520        LDY /SEED
032A- 20 F9 EA        1530        JSR LOAD.FAC.FROM.YA  $EAF9
                      1540 *---PREPARE SEED---------------
032D- A9 00           1550 .2     LDA #0        MAKE SEED POSITIVE
032F- 85 A2           1560        STA FAC.SIGN
0331- A5 9D           1570        LDA FAC       LIMIT SEED TO 67099547
0333- C9 9A           1580        CMP #$9A
0335- 90 07           1590        BCC .3
0337- A9 9A           1600        LDA #$9A
0339- 85 9D           1610        STA FAC
033B- 20 6E 03        1620        JSR MODULO
                      1630 *---(8192*SEED) MOD 67099547-----
033E- A9 0D           1640 .3     LDA #13
0340- 85 A5           1650        STA CNTR
0342- E6 9D           1660 .4     INC FAC
0344- 20 6E 03        1670        JSR MODULO
0347- C6 A5           1680        DEC CNTR
0349- D0 F7           1690        BNE .4
                      1700 *---SEED/67099547---------------
034B- A2 AE           1710        LDX #SEED
034D- A0 03           1720        LDY /SEED
034F- 20 2B EB        1730        JSR STORE.FAC.AT.YX.ROUNDED
0352- 20 66 EB        1740        JSR COPY.FAC.TO.ARG  $EB66
0355- A9 B3           1750 .5     LDA #FLT67
0357- A0 03           1760        LDY /FLT67
0359- 20 5C EA        1770        JSR FDIV.ARG.BY.YA $EA5C
                      1780 *---SCALE TEST-----------------
035C- AD A9 03        1790        LDA RANGE
035F- C9 82           1800        CMP #$82      IS RANGE BETWEEN ZERO AND ONE?
0361- 90 0A           1810        BCC .6        ...YES
                      1820 *---SCALE----------------------
0363- A9 A9           1830        LDA #RANGE
0365- A0 03           1840        LDY /RANGE
0367- 20 7F E9        1850        JSR FMUL.FAC.BY.YA    $E97F
036A- 20 23 EC        1860        JSR AS.INT   $EC23
                      1870 *---RETURN---------------------
036D- 60              1880 .6     RTS
                      1890 *-------------------------------
                      1900 MODULO
036E- A0 00           1910        LDY #0
0370- A5 9D           1920        LDA FAC
0372- C9 9A           1930        CMP #$9A
0374- 90 32           1940        BCC .3        < 67099547
0376- F0 02           1950        BEQ .1        67099547...
0378- A0 04           1960        LDY #4
037A- 38              1970 .1     SEC
037B- A5 A1           1980        LDA FAC+4     LSB
037D- F9 BB 03        1990        SBC MAN67+3,Y
0380- 48              2000        PHA
0381- A5 A0           2010        LDA FAC+3
0383- F9 BA 03        2020        SBC MAN67+2,Y
0386- 48              2030        PHA
0387- A5 9F           2040        LDA FAC+2
0389- F9 B9 03        2050        SBC MAN67+1,Y
038C- 48              2060        PHA
038D- A5 9E           2070        LDA FAC+1
038F- F9 B8 03        2080        SBC MAN67+0,Y
0392- 48              2090        PHA
0393- 90 0F           2100        BCC .2        <67099547
0395- 68              2110        PLA
0396- 85 9E           2120        STA FAC+1
0398- 68              2130        PLA
0399- 85 9F           2140        STA FAC+2
039B- 68              2150        PLA
039C- 85 A0           2160        STA FAC+3
039E- 68              2170        PLA
039F- 85 A1           2180        STA FAC+4
03A1- 4C 2E E8        2190        JMP NORMALIZE.FAC  $E82E
03A4- 68              2200 .2     PLA
03A5- 68              2210        PLA
03A6- 68              2220        PLA
03A7- 68              2230        PLA
03A8- 60              2240 .3     RTS
                      2250 *-------------------------------
```

```
03A9- 81 00 00
03AC- 00 00     2260 RANGE   .HS 81.00000000
03AE- 81 00 00
03B1- 00 00     2270 SEED    .HS 81.00000000
03B3- 9A 7F F6
03B6- E6 C0     2280 FLT67   .HS 9A.7FF6E6C0     = 67,099,547
03B8- FF F6 E6
03BB- C0        2290 MAN67   .HS FFF6E6C0
03BC- 7F FB 73
03BF- 60        2300         .HS 7FFB7360
                2310 *-------------------------------
```

The second listing is for my 32-bit algorithm based on Knuth's rules. Again, lines 1210-1270 set up the USR linkage. Lines 1360-1400 decide what kind of argument has been used. If negative, lines 1470-1590 prepare a new seed value. If zero, the previous value is re-used. If positive, the argument is the range.

In this version the seed is maintained as a 32-bit integer. Lines 1470-1590 convert from the floating point form of the argument in FAC to the integer form in SEED. If the argument happens to be bigger than 2^32, I simply force the exponent to 2^32.

Lines 1600-1690 form the next seed by multiplying by 314159269 and adding 907633386. The calculation is done in a somewhat tricky way. Essentially it involves loading 907633386 into the product register, and then adding the partial products of 314159269*seed to that register. The tricks allow me to do all that with a minimum of program and variable space, and I hope with plenty of speed. I understood it all this morning, but it is starting to get hazy now. If you really need a detailed explanation, call me some day. The modulo 2^32 part is automatic, because bits beyond 32 are thrown away.

Lines 1700-1780 load the seed value into FAC and convert it to a floating point fraction.

Lines 1790-1870 check the range requested. If less than 2, the fraction is returned as the USR result. If 2 or more, the fraction is multiplied by the range and integerized.

```
                1000 *-------------------------------
                1010 *SAVE S.RANDOM KNUTH
                1020 *-------------------------------
                1030 *      FROM KNUTH'S "THE ART OF COMPUTER PROGRAMMING"
                1040 *                   VOLUME 2, PAGES 155-157.
                1050 *-------------------------------
                1060        .OR $300
                1070        .TF B.RANDOM KNUTH
                1080 *-------------------------------
E82E-           1090 NORMALIZE.FAC       .EQ $E82E
E97F-           1100 FMUL.FAC.BY.YA      .EQ $E97F
EB2B-           1110 STORE.FAC.AT.YX.ROUNDED .EQ $EB2B
EBF2-           1120 AS.QINT             .EQ $EBF2
EC23-           1130 AS.INT              .EQ $EC23
                1140 *-------------------------------
000A-           1150 USER.VECTOR         .EQ $0A THRU $0C
009D-           1160 FAC                 .EQ $9D THRU $A2
00A2-           1170 FAC.SIGN            .EQ $A2
00AC-           1180 FAC.EXTENSION       .EQ $AC
00CA-           1190 AS.SEED             .EQ $CA THRU $CD
                1200 *-------------------------------
```

```
0300- A9 4C     1210 LINK   LDA #$4C     "JMP" OPCODE
0302- 85 0A     1220        STA USER.VECTOR
0304- A9 0D     1230        LDA #RANDOM
0306- 85 0B     1240        STA USER.VECTOR+1
0308- A9 03     1250        LDA /RANDOM
030A- 85 0C     1260        STA USER.VECTOR+2
030C- 60        1270        RTS
                1280 *-------------------------------
                1290 *      R = USR (X)
                1300 *      IF X < 0 THEN RESEED WITH ABS(X)
                1310 *      IF X = 0 THEN R = REPEAT OF PREVIOUS VALUE
                1320 *      IF 0 < X < 2 THEN GENERATE NEXT SEED AND RETURN
                1330 *                        0 <= R < 1
                1340 *      IF X >= 2 THEN R = INT(RND*X)
                1350 *-------------------------------
                1360 RANDOM
030D- A5 A2     1370        LDA FAC.SIGN CHECK FOR RESEEDING
030F- 30 0E     1380        BMI .1       ...YES
0311- A5 9D     1390        LDA FAC      CHECK FOR X=0
0313- F0 3B     1400        BEQ .6       ...YES, REUSE LAST NUMBER
                1410 *---X --> RANGE-----------------
0315- A2 A9     1420        LDX #RANGE
0317- A0 03     1430        LDY /RANGE
0319- 20 2B EB  1440        JSR STORE.FAC.AT.YX.ROUNDED   $EB2B
031C- 4C 3A 03  1450        JMP .4
                1460 *---PREPARE SEED----------------
031F- A9 00     1470 .1     LDA #0       MAKE SEED POSITIVE
0321- 85 A2     1480        STA FAC.SIGN
0323- A5 9D     1490        LDA FAC      LIMIT SEED TO 2^32-1
0325- C9 A0     1500        CMP #$A0
0327- 90 04     1510        BCC .2
0329- A9 A0     1520        LDA #$A0
032B- 85 9D     1530        STA FAC
032D- 20 F2 EB  1540 .2     JSR AS.QINT   $EBF2
0330- A2 03     1550        LDX #3       COPY FAC INTO SEED
0332- B5 9E     1560 .3     LDA FAC+1,X
0334- 9D AF 03  1570        STA SEED,X
0337- CA        1580        DEX
0338- 10 F8     1590        BPL .3
                1600 *---SEED*3141592692+907633386-----
033A- A2 00     1610 .4     LDX #0
033C- BD AF 03  1620 .5     LDA SEED,X
033F- 8D BC 03  1630        STA MULTIPLIER
0342- BD B8 03  1640        LDA C,X
0345- 9D AF 03  1650        STA SEED,X
0348- 20 73 03  1660        JSR MULTIPLY
034B- E8        1670        INX
034C- E0 04     1680        CPX #4
034E- 90 EC     1690        BCC .5
                1700 *---LOAD SEED INTO FAC-----------
0350- A2 05     1710 .6     LDX #5
0352- BD AE 03  1720 .7     LDA FLT.SEED,X
0355- 95 9D     1730        STA FAC,X
0357- CA        1740        DEX
0358- 10 F8     1750        BPL .7
035A- A9 00     1760        LDA #0
035C- 85 AC     1770        STA FAC.EXTENSION
035E- 20 2E E8  1780        JSR NORMALIZE.FAC
                1790 *---SCALE TEST------------------
0361- AD A9 03  1800        LDA RANGE
0364- C9 82     1810        CMP #$82     IS RANGE BETWEEN ZERO AND ONE?
0366- 90 0A     1820        BCC .8       ...YES
                1830 *---SCALE-----------------------
0368- A9 A9     1840        LDA #RANGE
036A- A0 03     1850        LDY /RANGE
036C- 20 7F E9  1860        JSR FMUL.FAC,BY.YA   $E97F
036F- 20 23 EC  1870        JSR AS.INT   $EC23
                1880 *---RETURN----------------------
0372- 60        1890 .8     RTS
                1900 *-------------------------------
                1910 MULTIPLY
0373- 8E C1 03  1920        STX BYTE.CNT
0376- A0 03     1930        LDY #3
0378- B9 B4 03  1940 .1     LDA A,Y
037B- 9D BD 03  1950        STA MULTIPLICAND,X
037E- 88        1960        DEY
037F- CA        1970        DEX
0380- 10 F6     1980        BPL .1
0382- A0 08     1990        LDY #8
0384- D0 07     2000        BNE .2       ...ALWAYS
```

```
                        2010 *--------------------------------
0386- 18                2020 .5     CLC           DOUBLE THE MULTIPLICAND
0387- 3E BD 03          2030 .6     ROL MULTIPLICAND,X
038A- CA                2040        DEX
038B- 10 FA             2050        BPL .6
038D- 4E BC 03          2060 .2     LSR MULTIPLIER
0390- 90 10             2070        BCC .4
0392- AE C1 03          2080        LDX BYTE.CNT
0395- 18                2090        CLC
0396- BD BD 03          2100 .3     LDA MULTIPLICAND,X
0399- 7D AF 03          2110        ADC SEED,X
039C- 9D AF 03          2120        STA SEED,X
039F- CA                2130        DEX
03A0- 10 F4             2140        BPL .3
03A2- AE C1 03          2150 .4     LDX BYTE.CNT
03A5- 88                2160        DEY
03A6- D0 DE             2170        BNE .5
03A8- 60                2180        RTS
                        2190 *--------------------------------
03A9- 81 00 00
03AC- 00 00             2200 RANGE          .HS 81.00000000
03AE- 80                2210 FLT.SEED       .HS 80
03AF- 00 00 00
03B2- 00                2220 SEED           .HS 00.00.00.00
03B3- 00                2230                .HS 00          SIGN
03B4- 12 B9 B0
03B7- A5                2240 A              .HS 12.B9.B0.A5   314159269
03B8- 36 19 62
03BB- EB                2250 C              .HS 36.19.62.EB   907633386
03BC-                   2260 MULTIPLIER     .BS 1
03BD-                   2270 MULTIPLICAND   .BS 4
03C1-                   2280 BYTE.CNT       .BS 1
                        2290 *--------------------------------
```

The third listing is cut down from the second one, to produce a 16-bit random number. The code is very similar to the program above, so I will not describe it line-by-line. If you want an optimized version of this, the multiply especially could be shortened.

```
                        1000 *--------------------------------
                        1010 *SAVE S.RANDOM KEYIN
                        1020 *--------------------------------
                        1030 *      ALLOWS ACCESS TO THE KEYIN RANDOM VALUE
                        1040 *--------------------------------
                        1050        .OR $300
                        1060 *      .TF B.RANDOM KEYIN
                        1070 *--------------------------------
E82E-                   1080 NORMALIZE.FAC      .EQ $E82E
E97F-                   1090 FMUL.FAC.BY.YA     .EQ $E97F
EB2B-                   1100 STORE.FAC.AT.YX.ROUNDED .EQ $EB2B
EBF2-                   1110 AS.QINT            .EQ $EBF2
EC23-                   1120 AS.INT             .EQ $EC23
                        1130 *--------------------------------
000A-                   1140 USER.VECTOR        .EQ $0A THRU $0C
009D-                   1150 FAC                .EQ $9D THRU $A2
00A2-                   1160 FAC.SIGN           .EQ $A2
00AC-                   1170 FAC.EXTENSION      .EQ $AC
004E-                   1180 KEY.SEED           .EQ $4E,4F
                        1190 *--------------------------------
0300- A9 4C             1200 LINK   LDA #$4C    "JMP" OPCODE
0302- 85 0A             1210        STA USER.VECTOR
0304- A9 0D             1220        LDA #RANDOM
0306- 85 0B             1230        STA USER.VECTOR+1
0308- A9 03             1240        LDA /RANDOM
030A- 85 0C             1250        STA USER.VECTOR+2
030C- 60                1260        RTS
                        1270 *--------------------------------
                        1280 *      R = USR (X)
                        1290 *      IF X < 0 THEN RESEED WITH ABS(X)
                        1300 *      IF X = 0 THEN R = REPEAT OF PREVIOUS VALUE
                        1310 *      IF 0 < X < 2 THEN GENERATE NEXT SEED AND RETURN
                        1320 *                        0 <= R < 1
                        1330 *      IF X >= 2 THEN R = INT(RND*X)
                        1340 *--------------------------------
```

```
                      1350 RANDOM
030D- A5 A2           1360         LDA FAC.SIGN CHECK FOR RESEEDING
030F- 30 0E           1370         BMI .1       ...YES
0311- A5 9D           1380         LDA FAC      CHECK FOR X=0
0313- F0 37           1390         BEQ .6       ...YES, REUSE LAST NUMBER
                      1400 *---X --> RANGE-----------------
0315- A2 AB           1410         LDX #RANGE
0317- A0 03           1420         LDY /RANGE
0319- 20 2B EB        1430         JSR STORE.FAC.AT.YX.ROUNDED   $EB2B
031C- 4C 38 03        1440         JMP .4
                      1450 *---PREPARE SEED----------------
031F- A9 00           1460 .1      LDA #0       MAKE SEED POSITIVE
0321- 85 A2           1470         STA FAC.SIGN
0323- A5 9D           1480         LDA FAC      LIMIT SEED TO 2^16-1
0325- C9 90           1490         CMP #$90
0327- 90 04           1500         BCC .2
0329- A9 90           1510         LDA #$90
032B- 85 9D           1520         STA FAC
032D- 20 F2 EB        1530 .2      JSR AS.QINT   $EBF2
0330- A5 A0           1540         LDA FAC+3
0332- 85 4E           1550         STA KEY.SEED
0334- A5 A1           1560         LDA FAC+4
0336- 85 4F           1570         STA KEY.SEED+1
                      1580 *---SEED*19125+13843------------
0338- A2 00           1590 .4      LDX #0
033A- B5 4E           1600 .5      LDA KEY.SEED,X
033C- 8D B4 03        1610         STA MULTIPLIER
033F- BD B2 03        1620         LDA C,X
0342- 95 4E           1630         STA KEY.SEED,X
0344- 20 77 03        1640         JSR MULTIPLY
0347- E8              1650         INX
0348- E0 02           1660         CPX #2
034A- 90 EE           1670         BCC .5
                      1680 *---LOAD SEED INTO FAC----------
034C- A9 00           1690 .6      LDA #0
034E- 85 A0           1700         STA FAC+3
0350- 85 A1           1710         STA FAC+4
0352- 85 A2           1720         STA FAC.SIGN
0354- 85 AC           1730         STA FAC.EXTENSION
0356- A9 80           1740         LDA #$80
0358- 85 9D           1750         STA FAC
035A- A5 4E           1760         LDA KEY.SEED
035C- 85 9E           1770         STA FAC+1
035E- A5 4F           1780         LDA KEY.SEED+1
0360- 85 9F           1790         STA FAC+2
0362- 20 2E E8        1800         JSR NORMALIZE.FAC
                      1810 *---SCALE TEST------------------
0365- AD AB 03        1820         LDA RANGE
0368- C9 82           1830         CMP #$82     IS RANGE BETWEEN ZERO AND ONE?
036A- 90 0A           1840         BCC .8       ...YES
```

```
                        1850 *---SCALE-----------------------
036C- A9 AB             1860         LDA #RANGE
036E- A0 03             1870         LDY /RANGE
0370- 20 7F E9          1880         JSR FMUL.FAC.BY.YA    $E97F
0373- 20 23 EC          1890         JSR AS.INT  $EC23
                        1900 *---RETURN----------------------
0376- 60                1910 .8      RTS
                        1920 *-------------------------------
                        1930 MULTIPLY
0377- 8E B7 03          1940         STX BYTE.CNT
037A- A0 01             1950         LDY #1
037C- B9 B0 03          1960 .1      LDA A,Y
037F- 9D B5 03          1970         STA MULTIPLICAND,X
0382- 88                1980         DEY
0383- CA                1990         DEX
0384- 10 F6             2000         BPL .1
0386- A0 08             2010         LDY #8
0388- D0 07             2020         BNE .2        ...ALWAYS
                        2030 *-------------------------------
038A- 18                2040 .5      CLC           DOUBLE THE MULTIPLICAND
038B- 3E B5 03          2050 .6      ROL MULTIPLICAND,X
038E- CA                2060         DEX
038F- 10 FA             2070         BPL .6
0391- 4E B4 03          2080 .2      LSR MULTIPLIER
0394- 90 0E             2090         BCC .4
0396- AE B7 03          2100         LDX BYTE.CNT
0399- 18                2110         CLC
039A- BD B5 03          2120 .3      LDA MULTIPLICAND,X
039D- 75 4E             2130         ADC KEY.SEED,X
039F- 95 4E             2140         STA KEY.SEED,X
03A1- CA                2150         DEX
03A2- 10 F6             2160         BPL .3
03A4- AE B7 03          2170 .4      LDX BYTE.CNT
03A7- 88                2180         DEY
03A8- D0 E0             2190         BNE .5
03AA- 60                2200         RTS
                        2210 *-------------------------------
03AB- 81 00 00
03AE- 00 00             2220 RANGE           .HS 81.00000000
03B0- 4A B5             2230 A               .DA /19125,#19125
03B2- 36 13             2240 C               .DA /13843,#13843
03B4-                   2250 MULTIPLIER      .BS 1
03B5-                   2260 MULTIPLICAND    .BS 2
03B7-                   2270 BYTE.CNT        .BS 1
                        2280 *-------------------------------
```

What do you do if you want even more randomness than you can get from one generator? You can use two together. The best way (for greatest randomness) is to use one to select values from a table produced by the other. First generate, say 50 or 100, random values with one generator. The generate a random value with the second generator and use it to pick one of the 50 or 100 values. That picked value is the first number to use. Then replace the picked value with a new value from the first generator. Pick another value randomly using the second generator, and so on. This is analogous to two people working together. The first person picks a bowlful at random from the universe. The second person picks items one at a time from the bowl. The first person keeps randomly picking from the universe to replace the items removed from the bowl by the second person.

You could use the 16-bit generator to pick values from a "bowl" kept full by my 32-bit generator.

Now back to those tests mentioned at the beginning. I am happy to report that all three of the algorithms listed above completely fill the hi-res screen, no holes left, eventually.

By the way, the August 1981 AAL contained an article about the Integer BASIC RND function, and how to use it from assembly language.

The Apple //c.....Bob Sander-Cederlof

In August 1977 I walked into CompuShop with checkbook in hand, hoping to fill a void in my life by (finally) buying my own personal computer. I didn't know one brand from another, but there was a 4K Apple II running a color demo in lo-res graphics that caught my eye. I bought it. My toy, because I certainly could think of no possible way to consider it more than a toy. The serial number is 219, and I am using it to write this article. By the way, the other brands that were at CompuShop in 1977 are now all out of business.

The price for 4K was $1298; I got 4K extra RAM and paid $1348 plus sales tax. No software. No CRT. No floating point BASIC. No slick manuals. About 45 pages of mimeographed notes was the total documentation package. I had to build a modulator kit that afternoon so I could hook it up to my TV set. The only other connection which seemed of any use was the cassette tape, which several hundred of you may remember. The store gave me a cassette containing the color demo and Woz's Breakout game. That was all there was! Eight empty slots, and absolutely nothing on the market to plug into them. Not even enough memory for hi-res graphics, which I did not even know existed. Absolutely no software for sale from any vendor.

I have spent a lot of time on this Apple. And money. And it is not JUST a toy any more! It has Applesoft on the motherboard, with 48K RAM. Slot 0 has an STB 128K RAM card (the best, in my opinion). All the other slots are full, but with what depends on the work for the day.

Now there is the Apple //c. $1295 buys you 128K RAM, Applesoft BASIC, a disk drive, and ProDOS! Probably over 10,000 programs on the market which will run in it, and many more to come. Built-in interfaces including two serial ports, mouse, disk controller, 80-columns, many video options, and more. The most often purchased interfaces are all there, enough to fill five slots in an older Apple. They added a headphone jack and volume control, too; it is recessed under the left edge. Using it will let you work later at night without disturbing light sleepers. You still get a "game" port, but it is a 9-pin D-socket and doubles as the mouse port. Sorry, no more Cassette port. A second disk drive can be added, and it costs significantly less than a second //e drive.

There are two new switches beside the RESET switch, labeled 40/80 and Keyboard. The first switches between 40 and 80 columns. The second selects QWERTY or Dvorak keyboard arrangement. Think a while of the implications to future generations of including THAT switch. The 40/80 switch is really just connected to what used to be cassette input $C060. You can read the switch position like the firmware does, by looking at the sign bit of that byte.

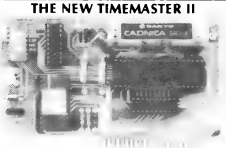
Until now all Apple game ports had four analog inputs, four switch outputs, and three switch inputs. The //c has only two analog inputs, and no switch outputs. The three switch inputs remain, with switch two dedicated to the mouse button. The

Apple Peripherals Are All We Make

That's Why We're So Good At It!

THE NEW TIMEMASTER II

Automatically date stamps files with PRO-DOS



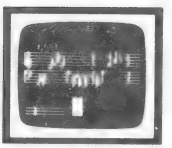
NEW 1984 DESIGN
An official PRO-DOS Clock

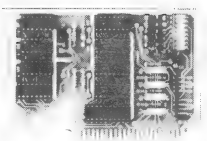
- Just plug it in and your programs can read the year, month, date, day, and time to 1 millisecond! The only clock with both year and ms.
- NiCad battery keeps the TIMEMASTER II running for over ten years.
- Full emulation of ALL other clocks. Yes, we emulate Brand A, Brand T, Brand P, Brand C, Brand S and Brand M too. It's easy for the TIMEMASTER to emulate other clocks, we just drop off features. That's why we can emulate others, but others CAN'T emulate us.
- The TIMEMASTER II will automatically emulate the correct clock card for the software you're using. You can also give the TIMEMASTER II a simple command to tell it which clock to emulate (but you'll like the Timemaster mode better). This is great for writing programs for those poor unfortunates that bought some other clock card.
- Basic, Machine Code, CP/M and Pascal software on 2 disks!
- Eight software controlled interrupts so you can execute two programs at the same time (many examples are included).
- On-board timer lets you time any interval up to 48 days long down to the nearest millisecond.

The TIMEMASTER II includes 2 disks with some really fantastic time oriented programs (over 40) including appointment book so you'll never forget to do anything again. Enter your appointments up to a year in advance then forget them. Plus DOS dater so it will automatically add the date when disk files are created or modified. The disk is over a $200.00 value alone– we give the software others sell. All software packages for business, data base management and communications are made to read the TIMEMASTER II. If you want the most powerful and the easiest to use clock for your Apple, you want a TIMEMASTER II

PRICE $129.00

Super Music Synthesizer
Improved Hardware and Software

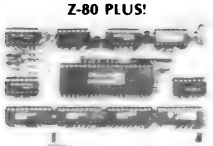




- Complete 16 voice music synthesizer on one card. Just plug it into your Apple, connect the audio cable (supplied) to your stereo, boot the disk supplied and you are ready to input and play songs.
- It's easy to program music with our compose software. You will start right away at inputting your favorite songs. The Hi-Res screen shows what you have entered in standard sheet music format.
- Now with new improved software for the easiest and the fastest music input system available anywhere.
- We give you lots of software. In addition to Compose and Play programs, 2 disks are filled with over 30 songs ready to play.
- Easy to program in Basic to generate complex sound effects. Now your games can have explosions, phaser zaps, train whistles, death cries. You name it, this card can do it.
- Four white noise generators which are great for sound effects.
- Plays music in true stereo as well as true discrete quadraphonic.
- Full control of attack, volume, decay, sustain and release.
- Will play songs written for ALF synthesizer (ALF software will not take advantage of all our card's features. Their software sounds the same in our synthesizer.)
- Our card will play notes from 30HZ to beyond human hearing.
- Automatic shutoff on power-up or if reset is pushed.
- Many many more features.

PRICE $159.00

Z-80 PLUS!



- TOTALLY compatible with ALL CP/M software.
- The only Z-80 card with a special 2K "CP/M detector" chip.
- Fully compatible with microsoft disks (no pre-boot required).
- Specifically designed for high speed operation in the Apple IIe (runs just as fast in the II+ and Franklin).
- Runs WORD STAR, dBASE II, COBOL-80, FORTRAN-80, PEACHTREE and ALL other CP/M software with no pre-boot.
- A semi-custom I.C. and a low parts count allows the Z-80 Plus to fly thru CP/M programs at a very low power level. (We use the Z-80A at fast 4MHZ.)
- Does EVERYTHING the other Z-80 boards do, plus Z-80 interrupts.

Don't confuse the Z-80 Plus with crude copies of the microsoft card. The Z-80 Plus employs a much more sophisticated and reliable design. With the Z-80 Plus you can access the largest body of software in existence. Two computers in one and the advantages of both, all at an unbelievably low price.

PRICE $139.00

Viewmaster 80

There used to be about a dozen 80 column cards for the Apple, now there's only ONE.

- TOTALLY Videx Compatible.
- 80 characters by 24 lines, with a sharp 7x9 dot matrix.
- On-board 40/80 soft video switch with manual 40 column override
- Fully compatible with ALL Apple languages and software—there are NO exceptions.
- Low power consumption through the use of CMOS devices.
- All connections are made with standard video connectors.
- Both upper and lower case characters are standard.
- All new design (using a new Microprocessor based C.R.T. controller) for a beautiful razor sharp display.
- The VIEWMASTER incorporates all the features of all other 80 column cards, plus many new improvements.

	PRICE	BUILT-IN SOFTSWITCH	SHIFT KEY SUPPORT	LOW POWER DESIGN	9X14 CUSTOM CHAR	7X9 DOT MATRIX	LIGHT PEN INPUTS	40 COLUMN OVERRIDE	INVERSE CHARACTERS
VIEWMASTER	179	YES	YES	YES	YES	YES	YES	YES	YES
SUPERTERM	MORE	NO	YES	NO	NO	NO	NO	YES	YES
WIZARD80	MORE	NO	NO	NO	NO	YES	NO	YES	YES
VISION80	MORE	YES	YES	NO	NO	YES	NO	NO	NO
OMNIVISION	MORE	NO	YES	NO	NO	NO	NO	YES	YES
VIEWMAX80	MORE	YES	YES	NO	NO	YES	NO	NO	YES
SMARTERM	MORE	YES	YES	NO	NO	NO	YES	YES	NO
VIDEOTERM	MORE	NO	NO	YES	NO	YES	YES	NO	YES

The VIEWMASTER 80 works with all 80 column applications including CP/M, Pascal, WordStar, Format II, Easywriter, Apple Writer II, VisiCalc, and all others. The VIEWMASTER 80 is THE MOST compatible 80 column card you can buy at ANY price!

PRICE $179.00

MemoryMaster IIe 128K RAM Card

- Expands your Apple IIe to 192K memory.
- Provides an 80 column text display.
- Compatible with all Apple IIe 80 column and extended 80 column card software (same physical size as Apple's 64K card).
- Can be used as a solid state disk drive to make your programs run up to 20 times FASTER (the 64K configuration will act as half a drive).
- Permits your IIe to use the new double high resolution graphics.
- Automatically expands Visicalc to 95 K storage in 80 columns! The 64K config. is all that's needed, 128K can take you even higher.
- PRO-DOS will use the MemoryMaster IIe as a high speed disk drive.
- Precision software disk emulation for Basic, Pascal and CP/M is available at a very low cost. NOT copy protected.
- Documentation included, we show you how to use all 192K.

If you already have Apple's 64K card, just order the MEMORYMASTER IIe with 64K and use the 64K from your old board to give you a full 128K. (The board is fully socketed so you simply plug in more chips.)

MemoryMaster IIe with 128K	**$249**
Upgradeable MemoryMaster IIe with 64K	**$169**
Non-Upgradeable MemoryMaster IIe with 64K	**$149**

Our boards are far superior to most of the consumer electronics made today. All I/C's are in high quality sockets with mil-spec. components used throughout. P.C. boards are glass-epoxy with gold contacts. Made in America to be the best in the world. All products work in the APPLE IIE, II, II+ and Franklin. The MemoryMaster IIe is IIe only. Applied Engineering also manufactures a full line of data acquisition and control products for the Apple: A/D converters and digital I/O cards, etc. Please call for more information. All our products are fully tested with complete documentation and available for immediate delivery. All products are guaranteed with a no hassle **THREE YEAR WARRANTY.**

Texas Residents Add 5% Sales Tax
Add $10.00 If Outside U.S.A.
Dealer Inquiries Welcome

Send Check or Money Order to:
APPLIED ENGINEERING
P.O. Box 798
Carrollton, TX 75006

Call (214) 492-2027
8 a.m. to 11 p.m. 7 days a week
MasterCard, Visa & C.O.D. Welcome
No extra charge for credit cards

other two analog input addresses are used as single bits to read the mouse X and Y direction. The four output bits are now used to control various interrrupt modes.

An interesting new softswitch input is at $C077. If bit 7 of the byte is 1, the current line being stroked on the screen is graphics; if 0, it is text. People like Bob Bishop, Don Lancaster, and Bill Budge probably already have figured out fantastic new tricks using this bit.

The power supply is now in a little box that is part of the power cord. 115 volts AC in, 12 volts DC out. The rest of the supply voltages derived inside the case. There will be a battery pack option later. And how about an adapter for running in the car?

The video output capability is phenomenal. Now you get all the American and European options built in. One connector gives you the NTSC we are all used to. Another gives you RF-modulated form for an American TV set. You also get RGB and various European standards. The 15-pin video connector also gives you an audio signal and various timing signals.

The ROM in the //c is VERY different. The differences include serial port and mouse firmware, better interrupt handling, the improvements made in the new //e ROMs, no more self-test program, and extensions to the disassembler (monitor L-command) for the 65C02 chip.

It is getting to be quite a chore for software to distinguish which kind of Apple II it is in. Here is a chart showing Apple's official ID bytes:

$FBB3	$FB1E	$FBC0	Environment
$38			Old (Original) Apple][
$EA	$AD		Apple][Plus Autostart
$EA	$8A		Apple /// Emulation
$06		$EA	Apple //e
$06		$E0	New Apple //e ROM
$06		$00	Apple //c

Interrupts are used extensively by the mouse firmware. A keyboard interrupt plus firmware implements a 128-character type-ahead buffer.

All this talk about mouse support leads me to make one clarification. You don't get a mouse unless you pay an extra $100. The firmware and interface are built-in, but the actual device is optional.

By the way, besides the 16 memory chips there are and only 21 other chips. More special chips, including IWM (Integrated Woz Machine, the disk controller); GLU (General Logical Unit); and TMG (Timing Generator). Compare the total 37 chips with about 50 in the Macintosh, and more than 90 in the IBM PCjr. Most of the chips are soldered in, but a few still sit in sockets.

Some Interesting News

Roger Wagner: well known to most of us as owner of Southwestern Data Systems, author of "Assembly Lines: The Book", author of several popular programs in early Apple days, speaker at AppleFests, and so forth. Roger is branching out.

Last month he incorporated and changed the name of SDS to Roger Wagner Publishing. Along with the name change, the product packaging has been changed. After a poll of dealers, they decided to replace the plush padded binders with a new package design which allows customers to browse through the manuals, while the diskette and other package contents are securely kept intact. No more shrink wrap! Simpler packaging is less expensive, so the prices of some products have been lowered. And one step further, even more significant: no more copy protection!

We applaud Roger for taking this step. As I remember it, Roger was one of the first publishers to use any kind of software protection, back in the 70's. His scheme included a program on the master disk which allowed you to make a limited number of back up copies. Now Roger joins us and a handful of other publishers who refuse to shackle users with protected software.

Roger has also joined forces with Val Golding (founder and long-time editor of Call-A.P.P.L.E.) to form Emerald City Publishing, Inc. Their first project is "The Apple's Apprentice", a magazine aimed at Apple-teens.

Apple //e ROM Revision.....Bob Sander-Cederlof

Dated March 21, 1984, I received a pair of 2764 ROMs and 12-page writeup. These are preliminary versions of a new set predicted to be in general distribution by early next year.

The new //e ROMs are substantially better than the current ones. Changes include:

Applesoft: modified to work in 80-column mode, and with lower case.

Monitor:

* modified to work with new Mouse ICON characters;

* modified to accept lowercase input;

* location $1F no longer used;

* miniassembler is back;

* search command added;

* IRQ handling substantially modified.

Video Firmware (after PR#3):

* fixed many bugs;

* no more jagged scrolling, now smooth and 30% faster;

* two new escape commands to enable/disable printing of control characters;

* SETVID ($FE93) now turns off 80-column mode;

* escape-R removed.

The new IRQ handler should finally make interrupts actually usable on the Apple. The old problem with location $45 is fixed. The settings of the various soft-switches which control memory mapping are saved and the machine is put into a cononical state. The standard IRQ return sequence will restore the interrupted state of all those switches.

The total overhead from IRQ-event to your IRQ-subroutine will run from 250 to 300 microseconds, depending on the soft-switch settings. If you are in a ProDOS environment, you will have to add all the overhead caused by ProDOS.

Of course, there will be new problems. ProDOS bent over backwards in a very strange way to solve the $45 problem with interrupts. Now that it is not necessary, ProDOS should be changed. But it can't be changed for the new and still work in the old, so.... The new IRQ and BRK handler also clobbers locations $100 and $101, which is BAD! Both those locations are used by Applesoft and many other programs!

If you think these changes will impact your work, or want to be involved in shaking out bugs, you might contact Developer Relations at Apple (408) 996-1010 and discuss the Certified Apple Developer program. I think it is because I am one of those that I received this material.

65C02 vs the older Apples.....Bob Sander-Cederlof

A few months ago we reported that apparently 2-MHz versions of the 65C02 chip worked in Apple IIs and II Plusses. (Even 1-MHz versions work in //e's.) Bob Stout was our source: he tried it, it worked, and he told us so.

Based on Bob's good luck, Stephen Bach tried it, it did not work, and he told us so. Steve and Bob got together, and it seems that the 2-MHz parts work in some IIs and II Plusses, but not all. "Try it and see" seems to be the only definitive answer.

By the way, you can get the 65C02 from Hamilton/Avnet and several other distributors for under $15 each. The 1MHz version is under $10 from Western Design Center. There is no incentive for dealers to get into the distribution of chips like this, because quantity price breaks depend on volumes in the thousands.

If you are having trouble finding a distributor, call Rockwell International's sales office; they might sell to you directly, point you to a distributor, or even give you a free sample. If not Rockwell, then try GTE or NCR, who also manufacture the 65C02, albeit without the extra 32 instructions Rockwell inserted. Here are some phone numbers for Rockwell:

California:	(714) 833-4655
Texas:	(214) 996-6500
Illinois:	(312) 297-8862
New Jersey:	(609) 596-0090
Tokyo:	(03) 265-8806
West Germany:	(089) 857-6016
England:	(01) 759-9911

You might possibly find these chips at Apple dealers or repair centers in the near future, because it is being used in the Apple //c. Apple is apparently not using the Rockwell version, because the BYTE article about the //c says the chip has 27 new opcodes. This is the total count of new opcodes including the new addressing modes added by the 65C02 offered by NCR, GTE, Western Design, and others. The Rockwell version adds an additonal 32. Those 32 are NOT in the 65802 or 65816, so chasing after them will lead you into dead-end streets.

If you are able to wait, the 65802 and 65816 far surpass the 65C02. You can order samples from Western Design Center, (602)962-4545, at $95 each. Originally expected in January, they are now targeting June 15th.

Decimal Floating Point Arithmetic.....Bob Sander-Cederlof

Perhaps you have wondered why PRINT (14.9 * 10) in Applesoft prints 148. This and many other such seeming bugs are a very common idiosyncrasy in the computer world.

Applesoft use binary floating point format for storing numbers and doing arithmetic. The number 14.9 is very clean in decimal, but it is an awful mess in binary. If you look at what is stored in RAM after doing X=14.9, you will find 84 6E 66 66 66. The first byte, 84, means the remaining four should be understood as four bits of binary integer (the "14" of "14.9") and 28 bits of binary fraction (the ".9" part). The first bit of the second byte is zero, which means the number is positive. Applesoft stores the sign in this bit position, knowing that ALL values other than 0.0 will have a 1-bit in this position of the magnitude.

Just before doing any arithmetic on the value above, Applesoft will unpack it, separating the sign, binary exponent, and the rest. The fancy name for the rest is the "mantissa". Writing out the mantissa for 14.9 we see EE 66 66 66. The first "E" means 14, and the .E666666 is APPROXIMATELY equal to .9. It is actual less than .9 by .000000066666666...forever. Since the number is not quite 14.9, multiplying by 10 gives not quite 149. And taking the INT of not-quite-149 gives the CORRECT answer of 148.

CORRECT, but not what you WANTED or EXPECTED. Right, Ethan? That is why you will find business software written in Applesoft is full of little fudge factors. We always need to multiply by enough 10's to make all pennies into integers, and then round up, and then truncate.

An alternative is to use DECIMAL arithmetic. And guess what: the 6502 has built-in decimal arithmetic. The only trouble is that Applesoft does not know about it.

I wrote an Applesoft extension package called DPFP which gives Applesoft 21-digit precision, rather than the normal 9. But it is still binary, so you still get those round-off and truncation problems with clearcut decimal fractions. About two and a half years ago I wrote another Applesoft extension package called DP18. This one is DECIMAL, and gives 18-digit precision. Bobby Deen helped me flesh it out with full support for arithmetic expressions and all the math functions.

Well, it has been hiding on my shelf long enough! I am going to start publishing it in AAL, a piece at a time. In this issue you will find the routines for addition and subtraction.

First a word about the way DP18 stores numbers. Since Applesoft uses five bytes for each floating point value, and since it is relatively easy to connect to Applesoft using multiples of five bytes, I use ten bytes for each DP18 value. The first byte holds the sign and exponent for the value. The remaining nine bytes hold 18 decimal digits, in BCD format. That is, each digit takes four bits.

The first bit of the first byte is the sign bit. Zero means plus, one means minus. If the whole first byte is zero, the whole number is zero. The remaining seven bits of the first byte are the decimal exponent, excess $40. The value $40 means ten to the zero power. $41 means 10, $42 means 100, and so on. $3F means .1, $3E means .01, and so on. Thus the exponent range is from $01 through $7F, meaning from 10^-63 through 10^64.

The mantissa bytes are considered to be a decimal fraction. The number is stored so that the most significant digit is always in the first nybble of the first byte, and the exponent is adjusted accordingly. Let's look at a few examples:

```
42 14 90 00 00 00 00 00 00 00  =  14.9
41 31 41 59 26 53 58 97 93 23  =  pi
38 50 00 00 00 00 00 00 00 00  =  .000000005
B8 50 00 00 00 00 00 00 00 00  =  -.000000005
```

Since listing the whole program at once is impossible, I have jumped right down to the lowest level so you can see how the elementary functions of addition and subtraction work. I put the origin at $0800 for this listing, but of course the final package will run wherever you assemble it for. Later we will get into I/O conversions, multiply and divide, math functions, print using, conversions between Applesoft and DP18 values, handling expressions with precedence and parentheses, and the linkage between DP18 and Applesoft.

The listing shown below has two main entry points, DSUB and DADD. You can guess what they mean! The two values to be operated on will already be unpacked into DAC and ARG by the time DSUB or DADD is called. Note that there is one extra byte for each accumulator, so that series of calculations will carry around an extra two digits of precision to avoid rounding errors. Unpacking a value into DAC involves storing the exponent byte in DAC.SIGN and then stripping the sign bit from DAC.EXPONENT.

DSUB and DADD both begin with the easiest cases, in which at least one of the values is zero. DSUB complements the value in DAC by merely toggling the sign bit, and then falls into DADD. In other words, ARG-DAC is the same as ARG+(-DAC).

DADD then determines which of the two values has the larger exponent. If necessary, it swaps ARG and DAC: the object is to have the value with the larger exponent in DAC (unless they are the same). Then the value in ARG is shifted right N digits, where N is the difference in the exponents. This what our teachers called "lining up the decimal points".

The subroutine which shifts ARG right N digits is rather smart. First, it will just fill ARG with zeros if the shift is 20 or more. Next, if the shift count is odd, it shifts right one digit position, or four bits. Then it does a direct move to shift the rest of the digits by N/2 bytes, and fills in with zero bytes on the left.

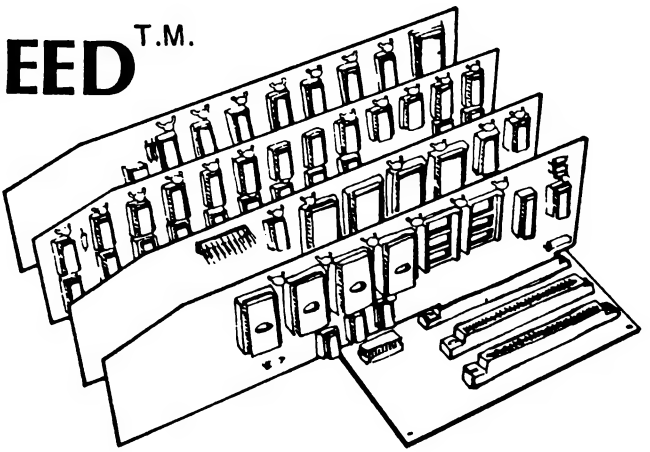
Addition is divided into two cases: either both arguments have the same sign, or they are different. If they are both the same, a simple addition loop is used. If the result carries into the next digit, DAC is shifted right one digit and a "1" is installed in the leftmost digit.

Otherwise, ARG is subtracted from DAC. If both ARG and DAC had the same exponents, it is possible that the value in ARG is larger than the value in DAC. In this case the subtracion loop will end with a "borrow" status, so the result needs to be complemented. I complement by subtracting from zero. Note that the three loops just described are all performed with the 6502 in decimal mode (the SED opcode at line 1490). CLD later reverts back to binary mode. After the mantissas are combined, the result may have one or more zero digits on the left. Therefore we go to a NORMALIZE subroutine.

NORMALIZE shifts the mantissa left until a non-zero digit is in the leftmost digit position. It also decrements the exponent for each digit-shift. I tried to do the shifting involved as intelligently as possible.

```
                    1000 *SAVE S.DP18 ADD & SUB
                    1010 *--------------------------------
                    1020 *       18-DIGIT DECIMAL FLOATING POINT
                    1030 *       ADDITION AND SUBTRACTION
                    1040 *--------------------------------
0800-               1050 DAC             .BS 12
0800-               1060 DAC.EXPONENT    .EQ DAC
0801-               1070 DAC.HI          .EQ DAC+1
080A-               1080 DAC.EXTENSION   .EQ DAC+10
080B-               1090 DAC.SIGN        .EQ DAC+11
                    1100 *--------------------------------
080C-               1110 ARG             .BS 12
080C-               1120 ARG.EXPONENT    .EQ ARG
080D-               1130 ARG.HI          .EQ ARG+1
0816-               1140 ARG.EXTENSION   .EQ ARG+10
0817-               1150 ARG.SIGN        .EQ ARG+11
                    1160 *--------------------------------
                    1170 SWAP.ARG.DAC
0818- A0 0B         1180         LDY #11       SWAP 12 BYTES
081A- B9 0C 08      1190 .1      LDA ARG,Y
081D- BE 00 08      1200         LDX DAC,Y
0820- 99 00 08      1210         STA DAC,Y
0823- 8A            1220         TXA
0824- 99 0C 08      1230         STA ARG,Y
0827- 88            1240         DEY
0828- 10 F0         1250         BPL .1
082A- 60            1260         RTS
                    1270 *--------------------------------
                    1280 *       SUBTRACT DAC FROM ARG
                    1290 * DAC = ARG - DAC
                    1300 *--------------------------------
082B- AD 00 08      1310 DSUB    LDA DAC.EXPONENT
082E- F0 E8         1320         BEQ SWAP.ARG.DAC  ARG-0=ARG
0830- AD 0B 08      1330         LDA DAC.SIGN
0833- 49 80         1340         EOR #$80
0835- 8D 0B 08      1350         STA DAC.SIGN
                    1360 *--------------------------------
                    1370 *       ADD ARG TO DAC
                    1380 * DAC = ARG + DAC
                    1390 *--------------------------------
0838- AD 0C 08      1400 DADD    LDA ARG.EXPONENT
083B- F0 34         1410         BEQ .3        DAC+0=DAC
083D- 38            1420 .1      SEC           COMPARE EXPONENTS
083E- AD 00 08      1430         LDA DAC.EXPONENT
0841- F0 D5         1440         BEQ SWAP.ARG.DAC  ARG+0=ARG
0843- ED 0C 08      1450         SBC ARG.EXPONENT
0846- 30 53         1460         BMI .8        ARG IS LARGER
0848- 20 C8 08      1470         JSR SHIFT.ARG.RIGHT.N
084B- F8            1480         SED           SET DECIMAL MODE
```

APPLESEED T.M.



Appleseed is a complete computer system. It is designed using the bus conventions established by Apple Computer for the Apple][. Appleseed is designed as an alternative to using a full Apple][computer system. The Appleseed product line includes more than a dozen items including CPU, RAM, EPROM, UART, UNIVERSAL Boards as well as a number of other compatible items. This ad will highlight the Mother board.

BX-DE-12 MOTHER BOARD

The BX-DE-12 Mother board is designed to be fully compatible with all of the Apple conventions. Ten card slots are provided. Seven of the slots are numbered in conformance with Apple standards. The additional three slots, lettered A, B and C, are used for boards which don't require a specific slot number. The CPU, RAM and EPROM boards are often placed in the slots A, B and C.

The main emphasis of the Appleseed system is illustrated by the Mother Board. The absolute minimum amount of circuitry is placed on the Mother Board; only the four ICs which are required for card slot selection are on the mother board. This approach helps in packaging (flexibility & smaller size), cost (buy only what you need) and repairability (isolate and fix problems through board substitution).

Appleseed products are made for O.E.M.s and serious industrial/scientific users. Send for literature on the full line of Appleseed products; and, watch here, each month, for additional items in the Appleseed line.

Appleseed products are not sold through computer stores.

Order direct from our plant in California.

Apple is a registered trademark of Apple Computer, Inc.

DOUGLAS ELECTRONICS
718 Marina Blvd., San Leandro, CA 94577 • (415) 483-8770

```
084C- AD 0B 08 1490          LDA DAC.SIGN COMPARE SIGNS
084F- 4D 17 08 1500          EOR ARG.SIGN
0852- 30 1E    1510          BMI .4       OPPOSITE SIGNS
               1520 *---SAME SIGNS----------------
0854- 18       1530          CLC          SAME SIGNS, JUST ADD VALUES
0855- A0 09    1540          LDY #9       TEN BYTES
0857- B9 01 08 1550 .2       LDA DAC.HI,Y
085A- 79 0D 08 1560          ADC ARG.HI,Y
085D- 99 01 08 1570          STA DAC.HI,Y
0860- 88       1580          DEY
0861- 10 F4    1590          BPL .2
0863- D8       1600          CLD          BINARY MODE
0864- 90 0B    1610          BCC .3       NO CARRY
0866- 20 A1 08 1620          JSR SHIFT.DAC.RIGHT.ONE
0869- AD 01 08 1630          LDA DAC.HI
086C- 09 10    1640          ORA #$10
086E- 8D 01 08 1650          STA DAC.HI
0871- 60       1660 .3       RTS
               1670 *---DIFFERENT SIGNS-----------
0872- 38       1680 .4       SEC          SUBTRACT ARG FROM FAC
0873- A0 09    1690          LDY #9       TEN BYTES
0875- B9 01 08 1700 .5       LDA DAC.HI,Y
0878- F9 0D 08 1710          SBC ARG.HI,Y
087B- 99 01 08 1720          STA DAC.HI,Y
087E- 88       1730          DEY
087F- 10 F4    1740          BPL .5
0881- B0 14    1750          BCS .7       NO BORROW
0883- 38       1760          SEC          BORROW, SO COMPLEMENT
0884- A0 09    1770          LDY #9
0886- A9 00    1780 .6       LDA #0
0888- F9 01 08 1790          SBC DAC.HI,Y
088B- 99 01 08 1800          STA DAC.HI,Y
088E- 88       1810          DEY
088F- 10 F5    1820          BPL .6
0891- AD 17 08 1830          LDA ARG.SIGN
0894- 8D 0B 08 1840          STA DAC.SIGN
0897- D8       1850 .7       CLD
0898- 4C F2 08 1860          JMP NORMALIZE.DAC
               1870 *---SWAP ARG & DAC, TRY AGAIN----
089B- 20 18 08 1880 .8       JSR SWAP.ARG.DAC
089E- 4C 3D 08 1890          JMP .1
               1900 *-------------------------------
               1910 *       SHIFT DAC RIGHT ONE DECIMAL DIGIT
               1920 *-------------------------------
               1930 SHIFT.DAC.RIGHT.ONE
08A1- EE 00 08 1940          INC DAC.EXPONENT
08A4- A0 04    1950          LDY #4       4 BITS RIGHT
08A6- 4E 01 08 1960 .1       LSR DAC.HI
08A9- 6E 02 08 1970          ROR DAC.HI+1
08AC- 6E 03 08 1980          ROR DAC.HI+2
08AF- 6E 04 08 1990          ROR DAC.HI+3
08B2- 6E 05 08 2000          ROR DAC.HI+4
08B5- 6E 06 08 2010          ROR DAC.HI+5
08B8- 6E 07 08 2020          ROR DAC.HI+6
08BB- 6E 08 08 2030          ROR DAC.HI+7
08BE- 6E 09 08 2040          ROR DAC.HI+8
08C1- 6E 0A 08 2050          ROR DAC.HI+9 EXTENSION
08C4- 88       2060          DEY
08C5- D0 DF    2070          BNE .1
08C7- 60       2080          RTS
               2090 *-------------------------------
               2100 *       SHIFT ARG RIGHT N DIGITS
               2110 *-------------------------------
               2120 SHIFT.ARG.RIGHT.N
08C8- A0 09    2130          LDY #9       SET UP FOR 10 BYTES
08CA- C9 14    2140          CMP #20      DON'T BOTHER IF OFF END
08CC- B0 1B    2150          BCS .4       JUST ENTER ZERO INTO ARG
08CE- 4A       2160          LSR          TEST SHIFT COUNT ODD OR EVEN
08CF- 90 03    2170          BCC .2       EVEN
08D1- 20 58 09 2180          JSR SHIFT.ARG.RIGHT.ONE
08D4- A8       2190 .2       TAY          # BYTES TO SHIFT
08D5- F0 1A    2200          BEQ .6       NONE
08D7- 49 FF    2210          EOR #$FF     -(#BYTES+1)
08D9- 18       2220          CLC
08DA- 69 0A    2230          ADC #10      9-#BYTES
08DC- AA       2240          TAX
08DD- A0 09    2250          LDY #9
08DF- BD 0D 08 2260 .3       LDA ARG.HI,X
08E2- 99 0D 08 2270          STA ARG.HI,Y
```

```
08E5- 88          2280          DEY
08E6- CA          2290          DEX
08E7- 10 F6       2300          BPL .3
08E9- A9 00       2310 .4       LDA #0
08EB- 99 0D 08    2320 .5       STA ARG.HI,Y
08EE- 88          2330          DEY
08EF- 10 FA       2340          BPL .5
08F1- 60          2350 .6       RTS
                  2360 *--------------------------------
                  2370 *       NORMALIZE VALUE IN DAC
                  2380 *--------------------------------
                  2390 NORMALIZE.DAC
08F2- A0 00       2400          LDY #0
08F4- B9 01 08    2410 .1       LDA DAC.HI,Y SEARCH FOR FIRST NON-ZERO BYTE
08F7- D0 0C       2420          BNE .2       FOUND IT
08F9- C8          2430          INY
08FA- C0 0A       2440          CPY #10
08FC- 90 F6       2450          BCC .1
08FE- 8D 00 08    2460          STA DAC.EXPONENT  NO NON-ZERO BYTES, SO
0901- 8D 0B 08    2470          STA DAC.SIGN      VALUE IS ZERO
0904- 60          2480          RTS
                  2490 *--------------------------------
0905- 98          2500 .2       TYA           TEST BYTE COUNT
0906- F0 1E       2510          BEQ .5        FIRST BYTE IS NON-ZERO
0908- A2 00       2520          LDX #0        POINT X AT FIRST BYTE
090A- B9 01 08    2530 .3       LDA DAC.HI,Y
090D- 9D 01 08    2540          STA DAC.HI,X
0910- E8          2550          INX
0911- C8          2560          INY
0912- C0 0A       2570          CPY #10
0914- 90 F4       2580          BCC .3
                  2590 *--------------------------------
0916- A9 00       2600          LDA #0        FILL REST OF DAC WITH ZEROES
0918- 9D 01 08    2610 .4       STA DAC.HI,X
091B- CE 00 08    2620          DEC DAC.EXPONENT  ADJUST EXPONENT
091E- CE 00 08    2630          DEC DAC.EXPONENT  FOR SHIFT DISTANCE
0921- E8          2640          INX
0922- E0 0A       2650          CPX #10
0924- 90 F2       2660          BCC .4
                  2670 *--------------------------------
0926- AD 01 08    2680 .5       LDA DAC.HI    SEE IF NEED ONE-DIGIT SHIFT
0929- 29 F0       2690          AND #$F0
092B- D0 06       2700          BNE .6        NO NYBBLE SHIFT NEEDED
092D- CE 00 08    2710          DEC DAC.EXPONENT
0930- 20 34 09    2720          JSR SHIFT.DAC.LEFT.ONE
0933- 60          2730 .6       RTS
                  2740 *--------------------------------
                  2750 SHIFT.DAC.LEFT.ONE
0934- A0 04       2760          LDY #4
0936- 0E 0A 08    2770 .1       ASL DAC.EXTENSION
0939- 2E 09 08    2780          ROL DAC.HI+8
093C- 2E 08 08    2790          ROL DAC.HI+7
093F- 2E 07 08    2800          ROL DAC.HI+6
0942- 2E 06 08    2810          ROL DAC.HI+5
0945- 2E 05 08    2820          ROL DAC.HI+4
0948- 2E 04 08    2830          ROL DAC.HI+3
094B- 2E 03 08    2840          ROL DAC.HI+2
094E- 2E 02 08    2850          ROL DAC.HI+1
0951- 2E 01 08    2860          ROL DAC.HI
0954- 88          2870          DEY
0955- D0 DF       2880          BNE .1
0957- 60          2890          RTS
                  2900 *--------------------------------
                  2910 *       SHIFT ARG RIGHT ONE DECIMAL DIGIT
                  2920 *--------------------------------
                  2930 SHIFT.ARG.RIGHT.ONE
0958- A0 04       2940          LDY #4
095A- 4E 0D 08    2950 .1       LSR ARG.HI
095D- 6E 0E 08    2960          ROR ARG.HI+1
0960- 6E 0F 08    2970          ROR ARG.HI+2
0963- 6E 10 08    2980          ROR ARG.HI+3
0966- 6E 11 08    2990          ROR ARG.HI+4
0969- 6E 12 08    3000          ROR ARG.HI+5
096C- 6E 13 08    3010          ROR ARG.HI+6
096F- 6E 14 08    3020          ROR ARG.HI+7
0972- 6E 15 08    3030          ROR ARG.HI+8
0975- 6E 16 08    3040          ROR ARG.HI+9  EXTENSION
0978- 88          3050          DEY
0979- D0 DF       3060          BNE .1
097B- 60          3070          RTS
```

What That Code Did.....Bob Sander-Cederlof

Way back in August 1981 I published a short article by John Broderick titled "What Does This Code Do?" Well, John never did tell us. But in the May 1984 Nibble, page 115, he finally has let the cat out of the bag. I think this article has probably been banging around the Nibble office for some time now, because John hasn't done anything with Apple's in quite a while. He developed a super fast accounting program in Apple II assembly language, then re-wrote the whole thing for the Sage 68000-based system. Last I heard he was in the IBM world.

The code he gave us three years ago was five bytes long:

```
        BRK
        PLA
        PLA
        PLA
        RTS
```

As published in Nibble, it is a little longer:

```
BREAK   BRK
        NOP
        PLA
        PLA
        JSR $FF3F
        RTS
```

Boiling it all down, John used this code during debugging sessions. By putting a JSR to the 8-byte program he can effect a clean breakpoint. Clean, in that he can use the monitor "G" command to continue execution after the BRK.

When JSR BREAK is executed, the BRK opcode will send Apple into the monitor and display the five registers. Their contents will have been saved at $45 thru $49. The address of the first PLA will also be saved. Typing the monitor "G" command will continue execution at that PLA. The two PLA's will pop off the return address the G command put on the stack, leaving it as it was before the BRK. The JSR $FF3F will restore the A-register, which the two PLA's clobbered. The the RTS will return right after the JSR BREAK which started this paragraph.

The original five-byte version was both confusing and erroneous. Confusing, because the PLA immediately after the BRK is never executed. BRK seems like a two-byte opcode to the 6502, so the saved address skips over the following byte. Erroneous, because the A-register has been changed by the time the RTS is executed. I think I would amend both of his versions to this:

```
BREAK   BRK
        NOP
        PLA
        PLA
        LDA $45
        RTS
```

Making a Map of Differences.....Bob Sander-Cederlof

Many times I have had two versions of the same program, and wondered where the differences might be.

For example, where are the differences between DOS 3.2 and 3.3, or between the various releases of DOS 3.3? And now that Apple has sent out some pre-releases of a new set of CDEF ROMs for the //e, where are the differences between these and the current //e ROMs?

I have always used the monitor V command to find them. By doing it a small piece at a time, I can pinpoint the changes. Then I turn on my printer and use the L command to document the new version wherever there are differences. But the piecemeal use of the V command wastes a lot of time. I wish I had some way of printing a complete map of all the differences....

What if I had a command which would compare two areas of memory, and print a map of differences? I could use a "." to represent matching locations, and a "*" to represent those that do not match. I could print either 32 or 64 per line: 32 on a 40-column screen, 64 on an 80-column screen or printer. Then I could tell at a glance where all changes had occurred!

I looked at the October 1981 issue of AAL to find out how to use the control-Y monitor command to add a new monitor feature. Then I looked in the listing of the monitor ROM (in my old "red" Apple Reference Manual) at the code for the V command and the command which prints a range of memory.

The program on the next page is the result.

Lines 1150-1190 set up the monitor control-Y vector. Booting DOS stores a branch which effectively makes the control-Y command do nothing. Storing the address of a real program there allows you to add your own commands to the monitor. Once installed, typing a control-Y into the monitor will execute the program named DIFFERENCES.

When we get there, if we typed a full length monitor command of the form "address1<address2.address3^Y" (by "^Y" I mean control-Y), all three of the addresses will have been converted to binary and stored in some standard locations. Address1 will be in $42 and $43, address2 in $3C and $3D, and address3 in $3E and $3F. We will interpret the addresses to mean to compare the block of memory beginning at address1 with the block running from address2 through address3.

Line 1220 prints a carriage return, the current address value in $3C and $3D, and a dash. Lines 1230-1280 compare the bytes at corresponding positions in the two blocks of memory, and select either a "." or a "*" accordingly. Line 1290 prints the selected character.

Lines 1300-1310 increment the two base addresses to point to the next byte in both memory blocks. The new address2 is also compared to address3 to see if we are finished yet.

Lines 1320-1350 check to see if we have printed all 32 on the current screen line. If not, back to .1 to print the next one. Otherwise, all the way back to print a new address and dash, starting a new line. If you want 64 bytes per line, change the mask in line 1330 from #$1F to #$3F. You might want to have the program check to see whether 80-columns is turned on or not, and automatically select #$1F or #$3F accordingly. You could also check to see if the output hook at $36, 37 is pointing at a printer, and use the longer lines.

Experiment. You'll learn a lot and have a lot of fun at the same time!

```
                      1000 *SAVE S.DIFFERENCES
                      1010 *-------------------------------
                      1020 *       DISPLAY MAP OF DIFFERENCES
                      1030 *       IN TWO MEMORY REGIONS
                      1040 *
                      1050 *       ADR1<ADR2.ADR3^Y
                      1060 *
                      1070 *-------------------------------
003C-                 1080 A1      .EQ $3C,3D
0042-                 1090 A4      .EQ $42,43
                      1100 *-------------------------------
FCB4-                 1110 MON.NXTA4  .EQ $FCB4
FD92-                 1120 MON.PRA1   .EQ $FD92
FDED-                 1130 MON.COUT   .EQ $FDED
                      1140 *-------------------------------
0800- A9 0B           1150 SETUP   LDA #DIFFERENCES
0802- 8D F9 03        1160         STA $3F9
0805- A9 08           1170         LDA /DIFFERENCES
0807- 8D FA 03        1180         STA $3FA
080A- 60              1190         RTS
                      1200 *-------------------------------
                      1210 DIFFERENCES
080B- 20 92 FD        1220         JSR MON.PRA1       PRINT CR, ADDRESS AND "-"
080E- A0 00           1230 .1      LDY #0             COMPARE TWO BYTES
0810- B1 3C           1240         LDA (A1),Y
0812- D1 42           1250         CMP (A4),Y
0814- F0 01           1260         BEQ .2             SAME, SELECT FIRST CHAR
0816- C8              1270         INY                DIFF, SELECT 2ND CHAR
0817- B9 2B 08        1280 .2      LDA CHARS,Y        GET DISPLAY CHAR
081A- 20 ED FD        1290         JSR MON.COUT       PRINT SAME OR DIFF CHAR
081D- 20 B4 FC        1300         JSR MON.NXTA4      NEXT ADDRESS AND TEST
0820- B0 08           1310         BCS .3             ...FINISHED
0822- A5 3C           1320         LDA A1             CHECK FOR FULL LINE
0824- 29 1F           1330         AND #$1F           OF 32
0826- D0 E6           1340         BNE .1             ...FULL YET
0828- F0 E1           1350         BEQ DIFFERENCES    ...FULL
082A- 60              1360 .3      RTS
                      1370 *-------------------------------
082B- AE AA           1380 CHARS   .AS -/.*/          SAME AND DIFF CHARS
                      1390 *-------------------------------
```

Apple Assembly Line is published monthly by S-C SOFTWARE CORPORATION, P.O. Box 280300, Dallas, Texas 75228. Phone (214) 324-2050. Subscription rate is $18 per year in the USA, sent Bulk Mail; add $3 for First Class postage in USA, Canada, and Mexico; add $12 postage for other countries. Back issues are available for $1.50 each (other countries add $1 per back issue for postage).